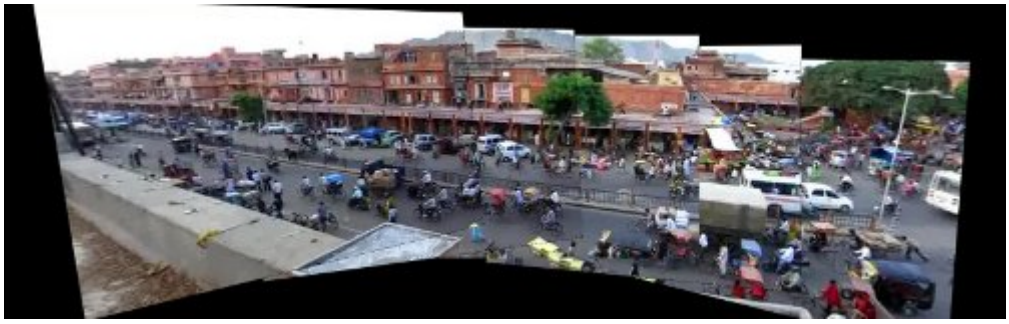
Programming Project #5: Video Stitching and Processing

CS445: Computational Photography

Due Date: 11:59pm on Tuesday, Nov. 12, 2019



In this project, you will experiment with interest points, image projection, and videos. You will manipulate videos by applying several transformations frame by frame. In doing so, you will explore correspondence using interest points, robust matching with RANSAC, homography, and background subtraction. You will also apply these techniques to videos by projecting and manipulating individual frames. You can also investigate cylindrical and spherical projection and other extensions of photo stitching and homography as bells and whistles.

In this project you can earn up to 230 points. We give you a starter kit that include most of the necessary files to start the project. Opencv-contrib is a library with various functions to extract and work with SIFT features. In this project -- unlike other projects -- you can use any standard numpy or scipy function.

The structure of starter package

In this project we provide a starter package that includes a video, individual frames in jpg format and some code to start. You can download the starter code here.

Opencv-contrib contains OpenCV's extra modules, among those a SIFT feature extraction/matching package. Before you start you need to initialize it with `sift = cv2.xfeatures2d.SIFT_create()`. The method `sift.detectAndCompute()` reads images and generates SIFT features. You can also use this function for your reference.

We provide a function named `auto_homography` that performs the steps of extracting SIFT features, matching, and setting up RANSAC. You are required to supply the homography code in the helper function `computeHomography`.

From OpenCV library you will extensively use `cv2.warpPerspective()` and optionally `cv2.perspectiveTransform()`. `cv2.warpPerspective()` inputs a 3x3 projection matrix (that auto_homography computes for you) and applies this transformation on an image that you provide. You will also need to use `dsize` argument to define the span of the output image that `imtransform.m` creates. Read OpenCV instructions for further details.

We give you a sequence of frames in jpg format so you don't need to worry about reading the video before you start. However, to produce video outputs for your report you need a software to convert your frames to a video.

Convert jpg sequence to video

If you have trouble with the Python code linked in Part 3 or want to extract frames from your own videos, we suggest installing and using ffmpeg on your computer. ffmpeg is a good command line library that can convert almost any video format to any other video format.

You can download ffmpeg for any OS here: https://www.ffmpeg.org/download.html. Using ffmpeg you can convert a set of jpg images to a video with the following command in Unix/Mac/Windows shell:

```
ffmpeg -r 30 -i aligned_frames/a%04d.jpg -pix_fmt yuv420p out.mp4
```

To convert a movie to a set of frames use:

```
ffmpeg -ss 00:00:00 -t 00:00:30 -i video.mp4 -r 30 frames/f%4d.jpg
```

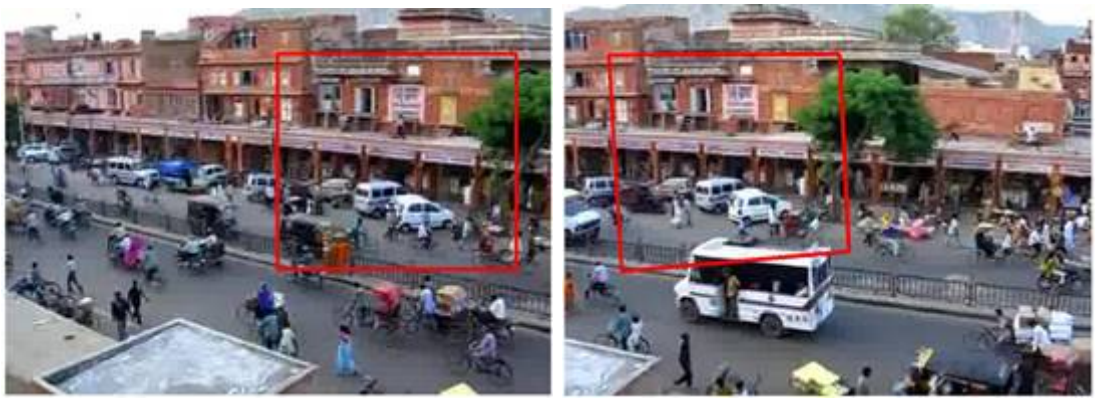
The argument -ss determines the starting point, -t determines the end point and -r determines the frame rate. After you produce your video outputs, upload them to Youtube and embed them in your report.

Part 1: Stitch two key frames [25 pts]

This is a video from 10 years ago in Jaipur, Northern India. We use the first 30 seconds that has 900 frames. Both the video and the decomposed jpg frames are included in the starter kit. We use frame number 450 as the reference frame. That means we map all other frames onto the "plane" of this frame using a homography transformation. This involves: (1) compute homography H between two frames; (2) project each frame onto the same surface; (3) blend the surfaces.

To stitch two overlapping video frames together, you first need to map one image plane to the other. To do that, you need to identify keypoints in both images, match between them to find point correspondences, and compute a projective transformation, called a homography that maps from one set of points to the other. Once you have recovered the homography, you can use it to project all frames onto the same coordinate space, and stitch them together to generate the video output. The provided `auto_homography.m` starter code (see description above) performs all the steps except for the homography computation. You may also want to experiment with the threshold used for RANSAC and recompute the homography based on all inliers after RANSAC.

Check that your homography is correct by plotting four points that form a square in frame 270 and their projections in each image, like this:



Include those images in your project page. Note that `cv2.warpPerspective()` takes output sizes, not coordinates, as input, i.e.:

```
img_warped = cv2.warpPerspective(img, H, (output_width, output_height))
```

The final warped image's pixels coordinate values need to be positive in order to be visualized. To evaluate this condition, apply the homography H to the original image's corner positions by a) `cv2.perspectiveTransform()` or b) the dot product (don't forget to divide the result by the last dimension). In case any warped image corners' coordinate value is negative, use these values to generate a translation matrix "H_t", then apply it to the homography "H" (obtained with auto_homography):

```
img_warped = cv2.warpPerspective(img, H_t.dot(H), (output_width, output_height))
```

You will need to write a blending function. You can use a simple method of replacing zero pixels in your output surface with non-zero pixels from each image. Better blending methods are bells and whistles. In this part you will map the frame number 270 on the reference image and produce an output like the following.

The images and videos in this project page are down-sampled but you need to produce a full resolution version.

Part 2: Panorama using five key frames [10 pts]

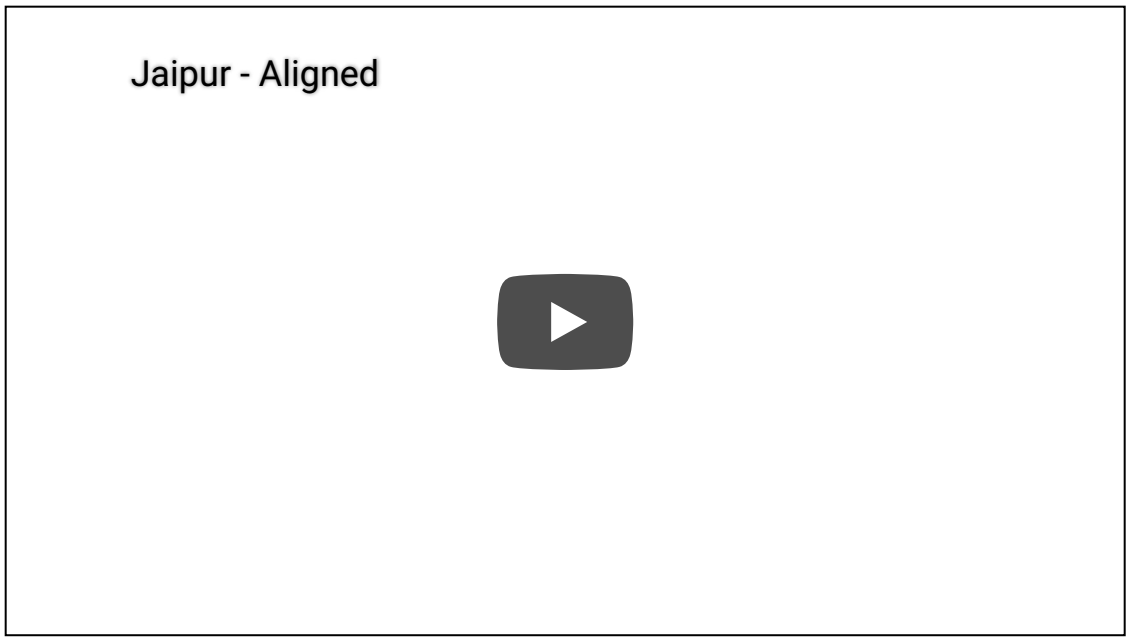
In this part you will produce a panorama using five key frames. Let's determine frames [90, 270, 450, 630, 810] as key frames. The goal is to map all the five frames onto the plane corresponding to frame 450 (that we also call the reference frame). For the frames 270 and 630 you can follow the instructions in part 1.

Mapping frame 90 to frame 450 is difficult because they share very little area. Therefore you need to perform a two stage mapping by using frame 270 as a guide. Compute one projection from 90 to 270 and one from 270 to 450 and multiply the two matrices. This produces a projection from 90 to 450 even though these frames have very little area in common.

For this stage, include your output panorama in your report.

Part 3: Map the video to the reference plane [15 pts]

In this part you will produce a video sequence by projecting all frames onto the plane corresponding to the reference frame (No. 450). For those frames that have small or no overlap with the reference frame you need to first map them onto the closest key frame. You can produce a direct homography between each frame and the reference frame by multiplying the two projection matrices. For this part output a video like the following:



To convert your individual frames to a video first save your frames in a separate folder and use utils.imageFolder2mpeg() to produce an output movie. Your frames could be named a0001.jpg to a0900.jpg. You can generate these names by using `cv2.imwrite('aligned_frames/a{:04d}.jpg'.format(i), i)` and then use `utils.imageFolder2mpeg('aligned_frames', fps=30)` to create the video. Upload your movie to youtube or a smiliar video hosting website and embed that in your report. If you have trouble with `imageFolder2mpeg`, follow the instructions for ffmpeg above.

Tips

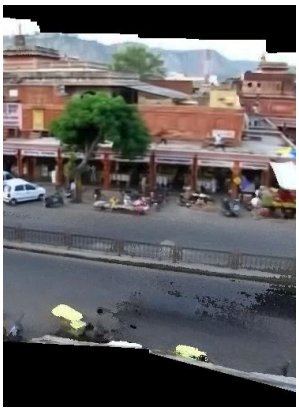
- This requires a lot of processing, so try processing and making a video from just 30 frames at first to make sure it works. When you process the whole sequence, it's a good time to get a coffee, as in take a leisurely stroll to your favorite coffee shop, enjoy your beverage, and saunter back in time to see your program finish.
- Save the homography from every frame. These may be useful later (and see tip below).
- Some default parameters have been set in the auto_homography code, including number of RANSAC iterations and inlier detection threshold. Also, the homography is re-estimated based on inliers. Feel free to change these parameters.
- Some frames may fail, either due to being unlucky with RANSAC (or not enough iterations), or you might consistently get a poor homography due to moving objects and difficulty in automatic matching to background points. If you save the homography for each frame, you can use any of the past frames as a reference. More recent frames will often provide better matches. You can fix mistakes frame-by-frame, or if you propose an automated solution to process all frames correctly without manual intervention, it may be worth extra points. Make sure to describe it in your writeup. There is some code in the utils.py file inside here for selecting frames that you may find helpful.

Part 4: Create background panorama [15 pts]

In this part you will remove moving objects from the video and create a background panorama that should incorporate pixels from all the frames.

In the video you produced in part 3 each pixel appears in several frames. You need to estimate which of the many colors correspond to the background. We take advantage of the fact that the background color is fixed while the foreground color changes frequently (because foreground moves). For example, a pixel on the street has a gray color. It can become red, green, white or black each for a short period of time. However, it appears gray more than any other color.

For each pixel in the sequence of part 3, determine all valid colors (colors that come from all frames that overlap that pixel). You can experiment with different methods for determining the background color of each pixel, as discussed in class. Perform the same procedure for all pixels and generate output. The output should be a completed panorama showing only pixels of background or non-moving objects.



As an example, to the right is a central portion of the background image produced by my code.

Part 5: Create background movie [10 pts]

Map the background panorama to the movie coordinates. For each frame of the movie, say frame 1, you need to estimate a projection from the panorama to frame 1. Note, you should be able to re-use the homographies that you estimated in Part 3. Perform this for all frames and generate a movie that looks like the input movie but shows only background pixels. All moving objects that belong to the foreground must be removed.

Part 6: Create foreground movie [15 pts]

In the background video, moving objects are removed. In each frame, those pixels that are different enough than the background color are considered foreground. For each frame determine foreground pixels and generate a movie that only includes foreground pixels.

Bells and whistles

Insert an unexpected object in the video [15 pts]

Add an unexpected object in the movie. Label the pixels in each frame as foreground or background. An inserted object must go below foreground and above background. Also note that an inserted object must appear fixed on the ground. Create a video that looks like original video with the tiny difference that some objects are inserted in the video.

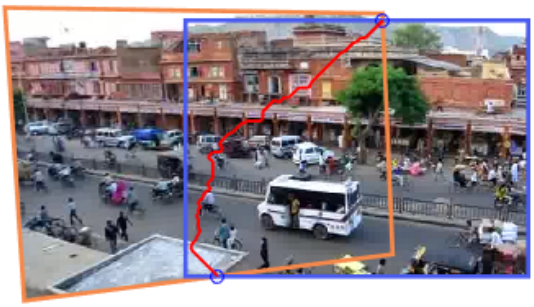
Process two more videos [up to 40 points]

You can apply the seven parts of the main project on two other videos. You get 20 points for processing one additional video and 40 points for processing two. If you do two additional videos, one of them must be your own. You get full points if you complete the first six parts of the main project.

Note that the camera position should not move in space but it can rotate. You also need to have some moving objects in the camera. Try to use your creativity to deliver something cool.

Smooth blending [up to 30 pts]

In part 2, you performed image tiling. In order to generate a good looking panorama, you need to select a seam line to cut through for each pair of individual images. Because the objects move the cut must be smart enough to avoid cutting through objects. You can also use Laplacian blending. You are free to use the code from your previous projects.



Generate a wide video [10 pts]

In Part 5 you created a background movie by projecting back the panorama background to each frame plane. If you map a wider area you will get a wider background movie. You can use this background movie to extend the borders of your video and make it wider. The extended video must be at least 50% wider. You can keep the same height.

Remove camera shake [20 pts]

You can track camera orientation using the homography matrices for each frame. This allows you to estimate and remove camera shake. Please note that camera shake removal for moving cameras is a more difficult problem and is an active area of research in computational photography. One idea (which we haven't tried and might not work) is to assume that camera parameters change smoothly and obtain a temporally smoothed estimate for each camera parameter. A better but more complicated method would be to solve for camera angle and focal length and smooth estimates for those parameters.

Make the street more crowded or a similar idea [15 pts]

You can use the techniques from the first bells and whistles task to add more people to the street. You can sample people from other frames that are a few seconds apart. You can alternatively show two copies of yourself in a video. Please note that your camera needs some rotation.

Deliverables

You know the drill: create a web page and thumbnail, and submit code/text/link on Compass. See project instructions for details.

Use words, images, and videos to show us what you've done.

- In each part, describe any special steps that you took or difficulties encountered.
- Include these images/videos:
  - Part 1: correspondence images and blended image. (25 pts)
  - Part 2: panorama from five key frames (10 pts)
  - Part 3: embedded video of mapping to reference frame (15 pts)
  - Part 4: panorama of background pixels (15 pts)
  - Part 5: video of background pixels (should look like input movie, but with foreground removed) (10 pts)
  - Part 6: video showing foreground pixels (complement to part 5) (15 pts)
- Bells&Whistles: include required results and description of how you obtained them

The core assignment is worth **100** points (including 10 points for quality of results/presentation). You can also earn up to **130 extra points** for the bells & whistles mentioned above.